IEICE TRANS. INF. & SYST., VOL.E98–D, NO.3 MARCH 2015

PAPER Special Section on Foundations of Computer Science—New Spirits in Theory of Computation and Algorithm—

Faster Enumeration of All Maximal Cliques in Unit Disk Graphs Using Geometric Structure

Taisuke IZUMI[†a)], *Member and* **Daisuke SUZUKI**[†*b)], *Nonmember*

SUMMARY This paper considers the problem of enumerating all maximal cliques in *unit disk graphs*, which is a plausible setting for applications of finding similar data groups. Our primary interest is to develop a faster algorithm using the geometric structure about the metric space where the input unit disk graph is embedded. Assuming that the distance between any two vertices is available, we propose a new algorithm based on two well-known algorithms called Bron-Kerbosch and Tomita-Tanaka-Takahashi. The key idea of our algorithm is to find a good pivot quickly using geometric proximity. We validate the practical impact of our algorithm via experimental evaluations.

key words: *enumerating maximal cliques, Bron-Kerbosch algorithm, Unit disk graph*

1. Introduction

Cliques are recognized as one of the most important structures in graph theory, and there are a large number of application areas demanding algorithms to find clique structures efficiently: Social network analysis [5], [13], bioinformatics [19], [20], data engineering [4], [25], computational topology [27], and so on. Some applications of those areas often require the list of all maximal cliques rather than a single large clique, which yields an interest to the problem of *enumeration of maximal cliques*. Since the number of maximal cliques can be exponential in the number of nodes, this problem trivially has no algorithm with a polynomial-time worst-case bound. However, many algorithms with reasonable running time in practice are currently known [6], [8], [9], [14]–[16], [21], [23]. Among them, the algorithm by Bron and Kerbosch [6] (referred to as BK algorithm hereafter) is the most famous one and has been used in many applications. Most of the recent advance about the maximal-clique enumeration is achieved by inventing a some modification of this algorithm, and our study also lies on the same line.

In this paper, we consider the problem of enumerating all maximal cliques in *unit disk graphs*, where each vertex is a point in a metric space and two vertices are connected with each other if they are within a unit distance. This problem setting is motivated by applications of finding similar groups in data: Each vertex represents one entry of the given data set and an edge represents a kind of similarity relationship between its endpoints. In such a scenario, each entry is often characterized by some numerical attributes. That is, it is a point in some multi-dimensional space. Then the similarity of two entries is often an interpretation of the proximity in the space. Hence the graph over the data set is typically a unit disk graph (in exact or approximated sense). This background brings us an interest to efficient algorithms making use of underlying geometric structures. The primary contribution of this paper is that we can actually have such an algorithm: Our starting point is the maximal-clique enumeration algorithm by Tomita et al. [16] (referred to as TTT algorithm hereafter), which is a modified version of BK algorithm and now is used in many application areas. The core of BK algorithm is the incremental constructions of maximal cliques via some recursive procedure. While it is simple and clever, it also produces the meaningless recursions which have no contribution to the output. So the technique called *pivoting* is often used. The pivoting technique prunes the branches of meaningless recursions by choosing a special node called *pivot* at each recursion step. The core of TTT algorithm is to select the best pivot with exhaustive search, which substantially reduces the total number of recursive calls. On the other hand, the cost of the pivot selection itself is relatively expensive.

In this paper, we show an alternative algorithm selecting good pivots for unit disk graphs. The geometric information the proposed algorithm requires is the distance between any two vertices, but the coordinate of each vertex is not necessary. Though its theoretical time bound is not so improved, by a careful implementation and several related heuristics, we can expect its practical running time is fast. We evaluate the proposed algorithm using random and real-data instances. The results show that our algorithm terminates substantially faster than TTT algorithm if the input graph is not so sparse. In addition, even for sparse instances, it is competitive to TTT one. So totally our algorithm achieves better performance

It should be noted that clique problems often become easier in unit disk graphs. For instance, the maximum clique problem is generally NP-complete but polynomially solvable in two-dimensional unit disk graphs [11]. In addition, the number of cliques in d-dimensional unit disk graphs of n vertices[**] is bounded by $O(n^d)$ because the number of cells in the arrangement of d-dimensional unit disks is bounded

Manuscript received April 2, 2014.
Manuscript revised August 4, 2014.

[†]The authors are with the Graduate School of Engineering, Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan.
[*]Presently with Denso Corporation.
a) E-mail: t-izumi@nitech.ac.jp
b) E-mail: cht15081@nitech.jp

DOI: 10.1587/transinf.2014FCP0018

[**]More precisely, in higher dimensional space, it might be more appropriate to say "unit ball (or sphere) graphs". However, in this paper we simply call them unit disk graphs.

by $O(n^d)$. That is, if d is a constant, the enumeration of maximal cliques in unit disk graphs is a task allowing a polynomial-time solution. However, even if d is not so large (e.g., three or four), this task is still practically expensive.

The organization of the paper is as follows: After stating the related work in Sect. 2, we first briefly explain Bron-Kerbosh algorithm and TTT algorithm in Sect. 3. At Sect. 4 we show our algorithm and its implementation details, where some heuristics improving the performance are also introduced. Section 5 gives the results of the experimental evaluation. Finally we conclude the paper at Sect. 6 with the note of future work.

2. Related Work

The maximal clique enumeration has a long history of research. A seminal paper initiating it is the one by Bron and Kerbosch [6]. Following this result, a number of algorithms has been proposed so far. As we mentioned in the introduction, since the output size (i.e., the number of maximal cliques) can become exponential in n, it is not meaningful to bound the total running time of the algorithms. Instead, the efficiency of algorithms is often evaluated by the delay (i.e., the worst-case computation time between two consecutive outputs). The algorithms achieving polynomial-time delay has been proposed in several papers [10], [17], [21], [23]. Another direction is the exploration of experimentally fast algorithms. The results on this line are Tomita et al. [16] and Eppstein et al. [14]. Algorithms running on some different computational models, such as parallel computation [22], and space-limited computation [7], [9], are also considered.

While all the results above handle the maximal clique enumeration for general graphs, several papers consider a restricted class of graphs, or slightly different problem settings. One of the popular related variants is the enumeration of bicliques in bipartite graphs. A number of algorithms are presented for this problem [1], [3], [12]. Another variant is the enumeration of pseudo cliques, which are the subgraphs with high edge density. An efficient algorithm for enumerating pseudo cliques are proposed by Uno [24]. On algorithms for restricted graph classes, several case are investigated: Bounded degeneracy graphs [14], odd-minor-free graphs [18], Power-law graphs [12], and so on. For unit disk graphs, there is no prior work explicitly considering the maximal clique enumeration, but the construction of Vietoris-Rips complex from a point set in any metric space [26], which is a fundamental problem in topological data analysis, is an equivalent problem.

3. Preliminaries

3.1 BK Algorithm

Since our method is based on BK algorithm, we first present its brief introduction. Let $G = (V, E)$ be an input graph. BK algorithm is defined as a recursive function taking three vertex subsets $R, P, X \subseteq V$ as arguments. The subset R is the vertex set of the clique currently constructed. The subset P consists of candidate vertices, which can be added to R for expanding the clique. The set X is the forbidden set of vertices. At each step of recursions, if both P and X are empty, it is guaranteed that R is maximal and thus it is outputted. Otherwise, the algorithm tries to expand the clique by adding a vertex in P. It first picks up and deletes one vertex v from P, and adds v to R. Then eliminating non-neighbors of v from P and X, the function is recursively executed. After the execution of the recursion, vertex v is moved from R to X. We show the pseudocode of BK algorithm in Algorithm 1. In the pseudocode, $N(v)$ denotes the set of v's neighbors (not including v itself).

Algorithm 1 BK Algorithm

```
1: BK(R, P, X)
2: if P and X are both empty then
3:     report R as a maximal clique
4: else
5:     for each vertex v in P do
6:         BK(R ∪ {v}, P ∩ N(v), X ∩ N(v))
7:         P := P \ {v}
8:         X := X ∪ {v}
9:     end for
10: end if
```

3.2 Pivoting

One problem in the original BK algorithm is that it often executes redundant recursive calls which do not report any maximal cliques. *Pivoting* is one of the promising techniques to reduce such recursive calls. The basic ideas of the pivoting are explained as follows: Assume that $BK(R, P, X)$ is called. Then, we choose an arbitrary vertex u from $P \cup X$ as the pivot. Obviously any maximal cliques including R must include the vertex that is not adjacent to u or u itself (otherwise it contradicts to the maximality). This fact implies that we do not have to call $BK(R \cup \{v\}, P \cap N(v), X \cap N(v))$ such that $v \in N(u)$ holds. Algorithm 2 shows the pseudocode of BK Algorithm with pivoting.

Algorithm 2 BK Algorithm with Pivoting

```
1: BK2(R, P, X)
2: if P and X are both empty then
3:     report R as a maximal clique
4: else
5:     choose a pivot vertex u in P ∪ X
6:     for each vertex v in P \ N(u) do
7:         BK2(R ∪ {v}, P ∩ N(v), X ∩ N(v))
8:         P := P \ {v}
9:         X := X ∪ {v}
10:    end for
11: end if
```

3.3 Pivot Selection Strategy by Tomita et al.

While we can use any vertex in $P \cup X$ as pivots, the running time of the algorithm drastically changes according to

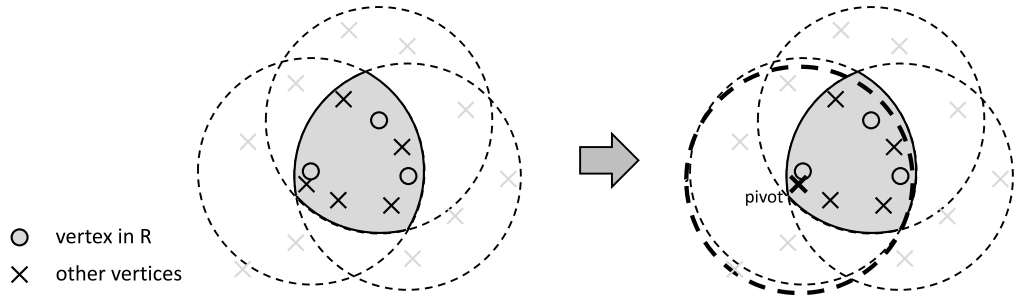

Fig. 1 An example of the pivot selection in unit disk graphs.

the selection of pivots. TTT algorithm is known as one of the best selection strategy. This strategy always chooses the vertex u maximizing $|P \cap N(u)|$ (in other words, minimizing the number of recursions called directly by the current recursion) as the pivot. The pseudocode of TTT algorithm is obtained by replacing line 5 in Algorithm 2 by the code shown in Algorithm 3.

Algorithm 3 TTT

```
1: for each u ∈ P ∪ X do
2:     count the u's neighbors in P
3: end for
4: select pivot u such that the number of neighbors in P is maximum
```

To find such vertex u, TTT algorithm exhaustively checks the size of $P \cap N(u)$ (in what follows, we call it the *score of* u) for each vertex u in $P \cup X$, which consumes $O(|P|\cdot(|P|+|X|))$ time if the adjacency matrix of input graphs is available (i.e. neighborhood relation is testable within a constant time). Since the other parts of TTT algorithm for one recursive call takes a time linear to $|P| + |X|$, this pivot selection strategy is relatively costly in particular when $|P|$ is large. Actually, in our preliminary experiments, we found that more than 80 percents of the running time was consumed for the pivot selection in TTT algorithm.

4. Our Algorithm

This section provides the details of our new algorithm. Its primary ingredient is a novel pivot-selection strategy utilizing the distance information between two vertices. Let $G = (V, E)$ be the input unit disk graph in any metric space. We assume that the distance $d(v, u)$ between any two vertices $u, v \in V$ is available and can be computed within a constant time.

4.1 Pivot Selection for Unit Disk Graphs

The fundamental idea of our algorithm is very simple. We consider the pivot selection in recursive call $BK(R, P, X)$. Letting $r \in R$ be an arbitrary vertex, any vertices in P must be covered by the unit disk centered at r. Then the unit disk centered at some vertex r' near to r is also expected to cover many points in P. If r' belongs to $P \cup X$, it seems to have a good score, and thus can be a good candidate of the pivot (Fig. 1). The core of our algorithm is to compute such a pivot candidate for each $r \in R$, and choose one with the highest score of all candidates. Algorithm 4 shows the pseudocode of our pivot selection strategy. More precisely, for each $r \in R$, the algorithm searches the vertex r' nearest to r in $P \cup X$, and computes its score by testing the adjacency between u and each node in P. After repeating this process for all vertices r, the vertex with the best score is chosen as the pivot. While a naive implementation of this algorithm takes $O(|P| + |X|)$ time for each $r \in R$ and thus totally $O(|R|(|P|+|X|))$ time is consumed for one pivot selection, we can improve its practical performance by a careful implementation explained in the next subsection.

Algorithm 4 Geometric Pivot Selection

```
1: maxintersect ← 0
2: for all r ∈ R do
3:     r' ← argmin_{p∈(P∪X)} d(r, p)
4:     if (|N(r') ∪ P| > maxintersect) then
5:         pivot ← r'
6:         maxintersect ← |N(r') ∪ P|
7:     end if
8: end for
```

4.2 Implementation Details

4.2.1 Some Observations

We explain the details of our implementation. To explain it, we first introduce several notations: Let us fix some pivot-selection strategy. Then we describe $(R_1, P_1, X_1) \prec (R_2, P_2, X_2)$ if and only if the execution of $BK(R_2, P_2, X_2)$ calls $BK(R_1, P_1, X_1)$. The score of the pivot selected by TTT algorithm for call $BK(R_1, P_1, X_1)$ is denoted by $ms(R_1, , P_1, X_1)$. Then, the following observations obviously hold.

Observation 1 $(R_1, P_1, X_1) \prec (R_2, P_2, X_2) \Rightarrow P_1 \cup X_1 \supseteq P_2 \cup X_2$

Observation 2 $(R_1, P_1, X_1) \prec (R_2, P_2, X_2) \Rightarrow ms(R_1, P_1, X_1) \geq ms(R_2, P_2, X_2)$

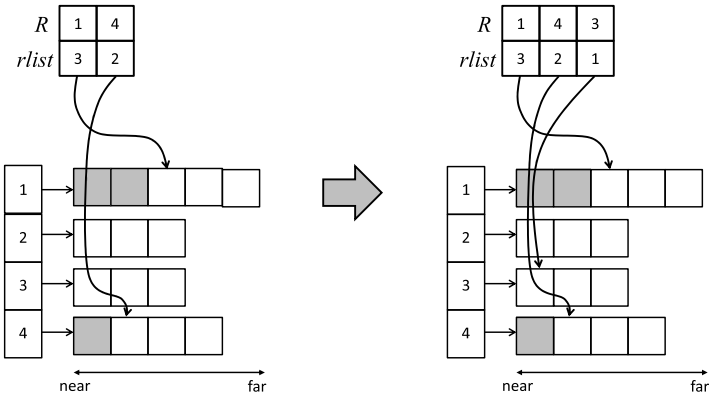

Fig. 2 Adding an element to R.

In the following argument we show a faster implementation of our algorithm and further heuristics based on these two observations.

4.2.2 Finding Nearest Candidates

In TTT algorithm, the input graph is normally given as the form of the adjacency matrix, and our algorithm also uses it. However, in addition to the adjacency matrix, our faster implementation utilizes a special form of its adjacency list, where the neighbor list for each vertex v is managed as the array sorted by the distance from v. We denote the array for vertex v by $A(v)$.

We also manage a pointer array *rlist* indexed by R. Each entry $rlist[r]$ for $r \in R$ is the pointer to some entry in $A(r)$. The array *rlist* is passed with set R when the recursive function is called (that is, in our implementation, BK algorithm takes four arguments, R, P, X, and *rlist*). When some new node v is added to R, a new entry corresponding to v is also added to *rlist*. The initial value $rlist[v]$ points the head of $A(v)$ (Fig. 2). At the beginning of some recursive call, $rlist[r]$ stores an index of $A(r)$ corresponding to the pivot candidate of r computed at the parent of the current recursion. The value of $rlist[r]$ is updated at the procedure of finding pivot candidates, shown as follows:

1. Check whether the element pointed by $rlist[r]$ is included in $P \cup X$ or not.
   - If included, take it as the pivot candidate for r and go to step 2.
   - Otherwise, increment the value of $rlist[r]$ and repeat step 1.
2. Calculate the score of the vertex pointed by $rlist[r]$.

Repeating this procedure for all $r \in R$, the algorithm completes both the pivot selection and the update of *rlist*. The reason why this procedure correctly returns pivot candidates can be understood as follows: From observation 1, the set $P \cup X$ monotonically grows up for the increase of the depth of recurrence. Thus, if $rlist[r] = k$ holds at the beginning of the current recursion, any node in $A(r)$ with index smaller than k is guaranteed not to be in $P \cup X$. It follows that the candidate nearest to r can be found by checking only the entries of $A(r)$ with index larger or equal to k.

By using this implementation, the choice of pivot candidates for each vertex in R becomes substantially fast. While the theoretical time bound is still $O(|R|(|P| + |X|))$, its typical running time behaves as $O(|R|)$. Thus the calculation of scores for each candidate is rather dominating part of our pivot selection strategy, which takes $O(|R||P|)$ time.

4.2.3 First Heuristic – Hybrid Approach

The asymptotic time estimation of our pivot-selection strategy raises up an interest to some hybrid approach with TTT strategy: As we stated, the asymptotic time bound for TTT strategy is $O(|P|(|P|+|X|))$ and for our algorithms, the bound is $O(|R|(|P| + |X|))$ and the realistic behavior is $O(|R||P|)$. Basically, as recurrence is deepened, R grows up and P goes small. Therefore, we can take the best of both the algorithms by switching them according to the value of $|P|$, $|X|$, and $|R|$. More precisely, this heuristic approach adopts our strategy if $|R| \leq \alpha(|P| + |X|)$, and TTT otherwise, where α is a design parameter biasing the choice of strategies.

4.2.4 Second Heuristic – Aborting Score Calculation

From observation 2, the score of the best pivot at each recursion is bounded by that of its parent recursion. It implies that the score of the pivot at the parent recursion can be used to measure how good the calculated score of a pivot candidate is. The principle of this heuristic is to abort the following score calculation once the algorithm finds a pivot with a good score competitive to the parent recursion. In this heuristic, the score s of the pivot at the parent recursion is passed as an argument of recursive calls. At the current recursion, if the algorithm finds a pivot candidate with score larger than or equal to $s - \beta$, it is immediately adopted as the pivot, where β is an "laziness" parameter.

5. Experiments

We implemented and evaluated the proposed algorithm for

Table 1 The results of the experimental evaluation.

nodes	dim.	$\bar{\delta}$	δ_{max}	Max Clique	Cliques	GPS(ms)	TTT(ms)	GPS/TTT
adult								
10000	3	83	282	109	60890	**9447**	14822	0.63
	5	69	271	108	50904	**9299**	12311	0.75
15000	3	149	464	174	196689	**36928**	101154	0.36
	5	123	439	173	161954	**46210**	63561	0.72
nomao								
10000	3	322	750	457	3336	**17848**	133477	0.13
	5	86	376	229	2831	**7266**	11461	0.63
15000	3	476	1065	710	4582	**54146**	over 10min	0.09
	5	126	488	318	3940	**17596**	45005	0.39
20000	5	197	757	493	13225	**44163**	253424	0.17
letter-recognition								
10000	3	78	182	183	470	**6013**	7053	0.85
	5	15	70	71	2359	**5175**	5288	0.98
15000	3	118	569	270	503	**14869**	18760	0.79
	5	22	106	107	2764	**13034**	13446	0.97
20000	3	158	377	378	530	**31668**	43579	0.73
	5	29	151	152	3089	**27580**	43477	0.63
Random								
10000	3	3	12	8	7513	5152	**5007**	1.02
15000	3	4	13	8	13098	14083	**13037**	1.08
20000	3	5	17	10	19634	**25602**	25838	0.99

several real-data and random instances. All of the following experiments are performed on the PC with Corei5-3210M CPU and 4GB memory. All algorithms are implemented by Java, and run on Java SE7 environment with JIT optimization.

The real data sets are obtained from the UCI machine learning repositories [2]. We choose three data sets for classification and clustering problems, called adult, nomao, and letter-recognition respectively. While some of those data sets have a high dimension and/or many data entries, we trimmed them into some fixed dimensions and sizes: three and five for dimensions, and 10000, 15000, and 20000 for the sizes of data sets. Every coordinate value is normalized into the range [0, 100] and the diameter of the unit disk is set at four in the standard L_2-metric. All random instances are obtained by placing a specified number of data points uniformly at random in the metric space. We ran our algorithm and TTT algorithm for all the prepared instances. Both of the two proposed heuristics are installed into our algorithm with design parameter $\alpha = 1.5$ and $\beta = 1$.

The result of the experiments is shown in Table 1. The columns #nodes, d, $\bar{\delta}$, δ_{max}, Max, and #Cliques respectively mean the number of nodes, dimension of the metric space, average degree, maximum degree, largest clique size, and total number of maximal cliques. GPS and TTT show the running times of our algorithm and TTT algorithm respectively. The last column GPS/TTT shows the running time ratio. Some of the results for instances with 20000 nodes are omitted because the algorithms do not complete by out-of-memory errors. Basically, the pivot selection in TTT algorithm becomes costly when the execution includes many recursive calls with large P. Since $|P|$ is clearly bounded by the degree of any node in R, TTT algorithm is expected to consume much time for instances with high average degrees and/or giant cliques. The experimental evaluation actually exhibits such tendency, and for those instances our algorithm is much (sometimes drastically) faster. On the other hand, TTT algorithm works efficiently for low average-degree instances, and thus we cannot find any strict advantage of our algorithm. However, even for such instances our algorithm is competitive to TTT, which implies that the overhead incurred by our algorithm is negligibly small. Totally, our algorithm achieves better performance for the overall experiments.

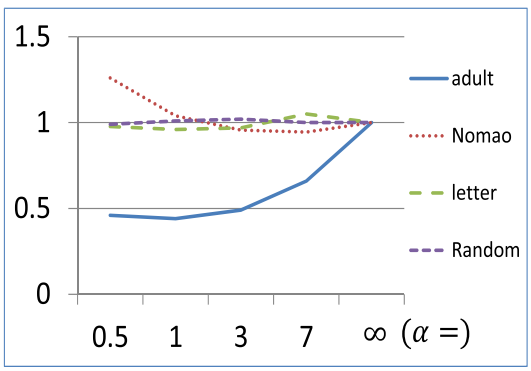

Fig. 3 The effect by the choice of α.

We also evaluated the effect of the design parameters α and β. For instances of dimensions five and 15000 vertices, we plot the performance of the proposed algorithm for value $\alpha = 0.5$, 1, 3, 7 and ∞, and $\beta = -\infty$, 1, 2 and 5. Note that $\alpha = \infty$ and $\beta = -\infty$ means that the algorithm does not use the corresponding heuristics. Figure 3 and 4 show the evaluation result, where the running time of the algorithm is plotted by the values relative to that for no use of the corresponding heuristics (i.e., $\alpha = \infty$ and $\beta = -\infty$). From the result, we can see two heuristics definitely improves the performance for the instances our algorithm efficiently solves.

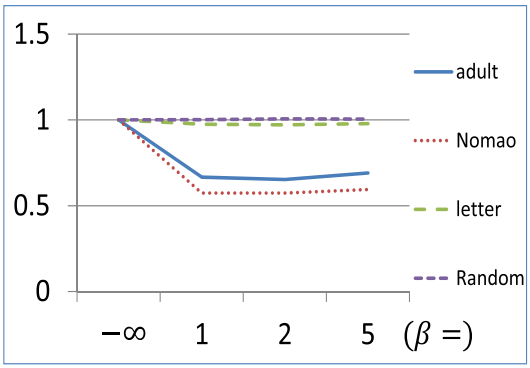

Fig. 4 The effect by the choice of β.

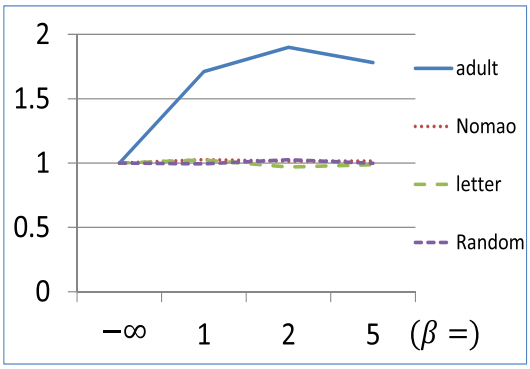

Fig. 5 The effect of the second heuristic applied to TTT algorithm.

About the choice of the parameters, $\alpha = 1.0 \sim 2$ and $\beta = 1$ stably marks good performance.

From this result, some readers may have the doubt that these heuristics are the dominant part of the speed-up in our algorithm. In fact, the second heuristic is independent of our geometric pivot selection, and thus applicable to the original TTT algorithm. So it looks plausible that the performance of TTT algorithm also considerably improves if we apply the second heuristic to it. However, interestingly it is not true. We present an result by the application of the second heuristics to TTT algorithm in Fig. 5. It shows that the abort heuristic is rather harmful for TTT algorithm.

6. Concluding Remarks

In this paper, we proposed a new algorithm for the problem of enumerating maximal cliques in unit disk graphs. The key ingredient of the proposed algorithm is the improvement of the running time by utilizing geometric information. Compared to TTT algorithm, our algorithm achieves better or competitive performance in practice, which is validated by the experimental evaluations for real-data and random instances.

An important future direction is the application of the proposed algorithm to massive instances. While our current implementation does not support the instances with an extremely large number of vertices, the recent result by Eppstein et al. [15] gives several techniques to make TTT algorithm applicable to large-sparse instances. The development of an algorithm reflecting those techniques and its evaluation is one of our future work. It is also an important problem to explore an algorithm to select better pivots much faster in unit disk graphs. While the paper is based on a simple closest-neighbor approach, , utilizing the smallest enclosing balls or convex hulls might lead a much better algorithm.

Acknowledgements

This work is supported in part by KAKENHI No.25106507 and No.25289114, and Inamori Foundation.

References

[1] A. Gély, L. Nourine, and B. Sadi, "Enumeration aspects of maximal cliques and bicliques," Discrete Appl. Math., vol.157, no.7, pp.1447–1459, 2009.
[2] The UC Irvine machine learning repository. http://archive.ics.uci.edu/ml/
[3] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P.L. Hammer, and B. Simeone, "Consensus algorithms for the generation of all maximal bicliques," Discrete Appl. Math., vol.145, no.1, pp.11–21, 2004.
[4] J.G. Augustson and J. Minker, An analysis of some graph theoretical cluster techniques, J. ACM, vol.17, no.4, pp.571–588, 1970.
[5] N.M. Berry, T.H. Ko, T. Moy, J. Smrcka, J. Turnley, and B. Wu, "Emergent clique formation in terrorist recruitment," Proc. AAAI-04 Workshop on Agent Organizations, 2004.
[6] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph," Commun. ACM, vol.16, no.9, pp.575–577, 1973.
[7] J. Cheng, Y. Ke, A.W.-C. Fu, J.X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by h*-graph," Proc. 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp.447–458, 2010.
[8] J. Cheng, Y. Ke, A.W.-C. Fu, J.X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," ACM Trans. Database Syst., vol.36, no.4, pp.21:1–21:34, 2011.
[9] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," Proc. 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.1240–1248, 2012.
[10] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," SIAM J. Comput., vol.14, no.1, pp.210–223, 1985.
[11] B.N. Clark, C.J. Colbourn, and D.S. Johnson, "Unit disk graphs," Discrete Mathematics, vol.86, no.1-3, pp.165–177, 1990.
[12] P. Damaschke, "Enumerating maximal bicliques in bipartite graphs with favorable degree sequences," Inf. Process. Lett., vol.114, no.6, pp.317–321, 2014.
[13] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu, "Community detection in large-scale social networks," Proc. 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, pp.16–25, 2007.
[14] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," Proc. 21st International Symposium on Algorithms and Computation (ISAAC), pp.403–414, 2010.
[15] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," Proc. 10th International Conference on Experimental algorithms (SEA), pp.364–375, 2011.
[16] E. ETomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," Theor. Comput. Sci., vol.363, no.1, pp.28–42, 2006.
[17] D.S. Johnson and C.H. Papadimitriou, "On generating all maximal independent sets," Inf. Process. Lett., vol.27, no.3, pp.119–123, 1988.

[18] K.-I. Kawarabayashi and D.R. Wood, "Cliques in odd-minor-free graphs," Proc. 18th Computing: the Australasian Theory Symposium (CATS), pp.133–138, 2012.
[19] I. Koch, "Enumerating all connected maximal common subgraphs in two graphs," Theor. Comput. Sci., vol.250, no.1-2, pp.1–30, 2001.
[20] I. Koch, T. Lengauer, and E. Wanke, "An algorithm for finding maximal common subtopologies in a set of protein structures," J. Computational Biology, vol.3, no.2, pp.289–306, 1996.
[21] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," 9th Scandinavian Workshop on Algorithm Theory (SWAT), pp.260–272, 2004.
[22] M.C. Schmidt, N.F. Samatova, K. Thomas, and B.-H. Park, "A scalable, parallel algorithm for maximal clique enumeration," J. Parallel and Distributed Computing, vol.69, no.4, pp.417–428, 2009.
[23] S. Tsukiyama, M. Ide, M. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all maximal independent sets," SIAM J. Comput., vol.6, no.3, pp.505–517, 1977.
[24] T. Uno, "An efficient algorithm for solving pseudo clique enumeration problem," Algorithmica, vol.56, no.1, pp.3–16, 2010.
[25] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," Proc. 3rd Internatilal Conference on Knowledge Discovery and Data Mining, pp.283–286, 1997.
[26] A. Zomorodian, "Fast construction of the vietoris-rips complex," Comput. & Graph., vol.34, no.3, pp.263–271, 2010.
[27] A. Zomorodian, "The tidy set: a minimal simplicial set for computing homology of clique complexes," Proc. Twenty-Sixth Annual Symposium on Computational Geometry (SoCG), pp.257–266, 2010.

Taisuke Izumi received the M.E. and D.I. degrees in computer science from Osaka University in 2003 and 2006. During 2006-2008, he worked at Nagoya Institute of Technology as an assistant professor. He is now an associate professor of Graduate School of Engineering, Nagoya Institute of Technology. His research interests include algorithms and distributed systems. He is a member of IEICE, ACM and IEEE.

Daisuke Suzuki received the B.E. and M.E. degrees in computer science from Nagoya Institute of Technology in 2012 and 2014. He works at Denso Corporation from 2014.